**Table and Index Organization**

**SQL Server 2008 R2**

[Other Versions](javascript:;)

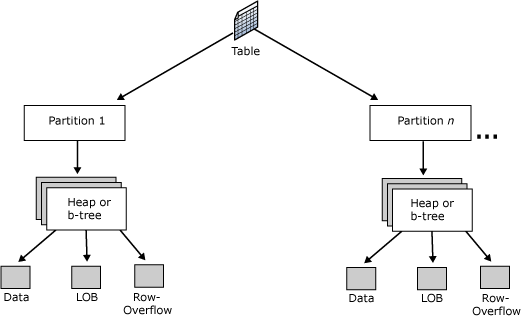
http://i.technet.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [SQL Server 2008](http://technet.microsoft.com/en-us/library/ms189051(d=printer,v=sql.100).aspx)
* [SQL Server 2005](http://technet.microsoft.com/en-us/library/ms189051(d=printer,v=sql.90).aspx)

Tables and indexes are stored as a collection of 8-KB pages. This topic describes the way table and index pages are organized.

[Table Organization](javascript:void(0))

The following illustration shows the organization of a table. A table is contained in one or more partitions and each partition contains data rows in either a heap or a clustered index structure. The pages of the heap or clustered index are managed in one or more allocation units, depending on the column types in the data rows.



[Partitions](javascript:void(0))

Table and index pages are contained in one or more partitions. A partition is a user-defined unit of data organization. By default, a table or index has only one partition that contains all the table or index pages. The partition resides in a single filegroup. A table or index with a single partition is equivalent to the organizational structure of tables and indexes in earlier versions of SQL Server.

When a table or index uses multiple partitions, the data is partitioned horizontally so that groups of rows are mapped into individual partitions, based on a specified column. The partitions can be put on one or more filegroups in the database. The table or index is treated as a single logical entity when queries or updates are performed on the data. For more information, see [Partitioned Tables and Indexes](http://technet.microsoft.com/en-us/library/ms188706(v=sql.105).aspx).

To view the partitions used by a table or index, use the [sys.partitions (Transact-SQL)](http://technet.microsoft.com/en-us/library/ms175012(v=sql.105).aspx) catalog view.

[Clustered Tables, Heaps, and Indexes](javascript:void(0))

SQL Server tables use one of two methods to organize their data pages within a partition:

* Clustered tables are tables that have a clustered index.

The data rows are stored in order based on the clustered index key. The clustered index is implemented as a B-tree index structure that supports fast retrieval of the rows, based on their clustered index key values. The pages in each level of the index, including the data pages in the leaf level, are linked in a doubly-linked list. However, navigation from one level to another is performed by using key values. For more information, see [Clustered Index Structures](http://technet.microsoft.com/en-us/library/ms177443(v=sql.105).aspx).

* Heaps are tables that have no clustered index.

The data rows are not stored in any particular order, and there is no particular order to the sequence of the data pages. The data pages are not linked in a linked list. For more information, see [Heap Structures](http://technet.microsoft.com/en-us/library/ms188270(v=sql.105).aspx).

Indexed views have the same storage structure as clustered tables.

When a heap or a clustered table has multiple partitions, each partition has a heap or B-tree structure that contains the group of rows for that specific partition. For example, if a clustered table has four partitions, there are four B-trees; one in each partition.

**Nonclustered Indexes**

Nonclustered indexes have a B-tree index structure similar to the one in clustered indexes. The difference is that nonclustered indexes do not affect the order of the data rows. The leaf level contains index rows. Each index row contains the nonclustered key value, a row locator and any included, or nonkey, columns. The locator points to the data row that has the key value. For more information, see [Nonclustered Index Structures](http://technet.microsoft.com/en-us/library/ms177484(v=sql.105).aspx).

**XML Indexes**

One primary and several secondary XML indexes can be created on each xml column in the table. An XML index is a shredded and persisted representation of the XML binary large objects (BLOBs) in the xml data type column. XML indexes are stored as internal tables. To view information about xml indexes, use the [sys.xml\_indexes](http://technet.microsoft.com/en-us/library/ms176003(v=sql.105).aspx) or [sys.internal\_tables](http://technet.microsoft.com/en-us/library/ms187799(v=sql.105).aspx) catalog views.

For more information about XML indexes, see [Indexes on XML Data Type Columns](http://technet.microsoft.com/en-us/library/ms191497(v=sql.105).aspx).

[Allocation Units](javascript:void(0))

An allocation unit is a collection of pages within a heap or B-tree used to manage data based on their page type. The following table lists the types of allocation units used to manage data in tables and indexes.

|  |  |
| --- | --- |
| **Allocation unit type** | **Is used to manage** |
| IN\_ROW\_DATA | Data or index rows that contain all data, except large object (LOB) data.  Pages are of type Data or Index. |
| LOB\_DATA | Large object data stored in one or more of these data types: text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max), or CLR user-defined types (CLR UDT).  Pages are of type Text/Image. |
| ROW\_OVERFLOW\_DATA | Variable length data stored in varchar, nvarchar, varbinary, or sql\_variant columns that exceed the 8,060 byte row size limit.  Pages are of type Text/Image. |

For more information about page types, see [Understanding Pages and Extents](http://technet.microsoft.com/en-us/library/ms190969(v=sql.105).aspx).

A heap or B-tree can have only one allocation unit of each type in a specific partition. To view the table or index allocation unit information, use the [sys.allocation\_units](http://technet.microsoft.com/en-us/library/ms189792(v=sql.105).aspx) catalog view.

**IN\_ROW\_DATA Allocation Unit**

For every partition used by a table (heap or clustered table), index, or indexed view, there is one IN\_ROW\_DATA allocation unit that is made up of a collection of data pages. This allocation unit also contains additional collections of pages to implement each nonclustered and XML index defined for the table or view. The page collections in each partition of a table, index, or indexed view are anchored by page pointers in the **sys.system\_internals\_allocation\_units** system view.

|  |
| --- |
| **Important noteImportant** |
| The **sys.system\_internals\_allocation\_units** system view is reserved for Microsoft SQL Server internal use only. Future compatibility not guaranteed. |

Each table, index, and indexed view partition has a row in **sys.system\_internals\_allocation\_units** uniquely identified by a container ID (**container\_id**). The container ID has a one-to-one mapping to the **partition\_id** in the **sys.partitions** catalog view that maintains the relationship between the table, index, or the indexed view data stored in a partition and the allocation units used to manage the data within the partition.

The allocation of pages to a table, index, or an indexed view partition is managed by a chain of IAM pages. The column **first\_iam\_page** in **sys.system\_internals\_allocation\_units** points to the first IAM page in the chain of IAM pages managing the space allocated to the table, index, or the indexed view in the IN\_ROW\_DATA allocation unit.

**sys.partitions** returns a row for each partition in a table or index.

* A heap has a row in **sys.partitions** with **index\_id** = 0.

The **first\_iam\_page** column in **sys.system\_internals\_allocation\_units** points to the IAM chain for the collection of heap data pages in the specified partition. The server uses the IAM pages to find the pages in the data page collection, because they are not linked.

* A clustered index on a table or a view has a row in **sys.partitions** with **index\_id** = 1.

The **root\_page** column in **sys.system\_internals\_allocation\_units** points to the top of the clustered index B-tree in the specified partition. The server uses the index B-tree to find the data pages in the partition.

* Each nonclustered index created for a table or a view has a row in **sys.partitions** with **index\_id** > 1.

The **root\_page** column in **sys.system\_internals\_allocation\_units** points to the top of the nonclustered index B-tree in the specified partition.

* Each table that has at least one LOB column also has a row in **sys.partitions** with **index\_id** > 250.

The **first\_iam\_page** column points to the chain of IAM pages that manage the pages in the LOB\_DATA allocation unit.

**ROW\_OVERFLOW\_DATA Allocation Unit**

For every partition used by a table (heap or clustered table), index, or indexed view, there is one ROW\_OVERFLOW\_DATA allocation unit. This allocation unit contains zero (0) pages until a data row with variable length columns (varchar, nvarchar, varbinary, or sql\_variant) in the IN\_ROW\_DATA allocation unit exceeds the 8 KB row size limit. When the size limitation is reached, SQL Server moves the column with the largest width from that row to a page in the ROW\_OVERFLOW\_DATA allocation unit. A 24-byte pointer to this off-row data is maintained on the original page.

Text/Image pages in the ROW\_OVERFLOW\_DATA allocation unit are managed in the same way pages in the LOB\_DATA allocation unit are managed. That is, the Text/Image pages are managed by a chain of IAM pages.

**LOB\_DATA Allocation Unit**

When a table or index has one or more LOB data types, one LOB\_DATA allocation unit per partition is allocated to manage the storage of that data. The LOB data types include text, ntext, image, xml, varchar(max), nvarchar(max), varbinary(max), and CLR user-defined types.

[Partition and Allocation Unit Example](javascript:void(0))

The following example returns partition and allocation unit data for two tables: DatabaseLog, a heap with LOB data and no nonclustered indexes, and Currency, a clustered table without LOB data and one nonclustered index. Both tables have a single partition.

USE AdventureWorks2008R2;

GO

SELECT o.name AS table\_name,p.index\_id, i.name AS index\_name , au.type\_desc AS allocation\_type, au.data\_pages, partition\_number

FROM sys.allocation\_units AS au

JOIN sys.partitions AS p ON au.container\_id = p.partition\_id

JOIN sys.objects AS o ON p.object\_id = o.object\_id

JOIN sys.indexes AS i ON p.index\_id = i.index\_id AND i.object\_id = p.object\_id

WHERE o.name = N'DatabaseLog' OR o.name = N'Currency'

ORDER BY o.name, p.index\_id;

Here is the result set. Notice that the DatabaseLog table uses all three allocation unit types, because it contains both data and Text/Image page types. The Currency table does not have LOB data, but does have the allocation unit required to manage data pages. If the Currency table is later modified to include a LOB data type column, a LOB\_DATA allocation unit is created to manage that data.

table\_name index\_id index\_name allocation\_type data\_pages partition\_number

----------- -------- ----------------------- --------------- ----------- ------------

Currency 1 PK\_Currency\_CurrencyCode IN\_ROW\_DATA 1 1

Currency 3 AK\_Currency\_Name IN\_ROW\_DATA 1 1

DatabaseLog 0 NULL IN\_ROW\_DATA 160 1

DatabaseLog 0 NULL ROW\_OVERFLOW\_DATA 0 1

DatabaseLog 0 NULL LOB\_DATA 49 1

(5 row(s) affected)